First Hit Fwd Refs

☐ Generate Collection Print

L4: Entry 4 of 16 File: USPT Apr 1, 2003

DOCUMENT-IDENTIFIER: US 6542967 B1
TITLE: Cache object store

Brief Summary Text (6):
Two common paradigms for the persistent storage of data are file systems and database systems. A file system contains general knowledge of the organization of the data stored on storage devices, such as memories and disks, needed to implement properties/performance of a desired storage architecture. A database system provides as structured data model that may be implemented on a file system or other storage architecture. Notably, there is an expectancy that the data (i.e., "content") stored on the file system or database will be preserved until explicitly removed. Persistency with respect to the storage of content, e.g., naming of data files and their non-volative storage, is paramount to other properties/performance metrics such as organization of, and speed of access to, the to stored content. As such, these characteristics of a file system or database are not generally suited to the access and volatility characteristics of a cache system.

Brief Summary Text (11):
Yet another feature of the invention is to provide multiple memory abstractions that allow exploitation of the characteristics of a cache environment.

Detailed Description Text (9):
FIG. 3 is a schematic block diagram of the enhanced cache system which is preferably organized as a novel cache object store 300 to provide fast and efficient storage of data as cache objects 302. A cache object is characterized as a collection of data that is persistent over a predetermined period of time, but that can be recovered if lost. For example, if a user request is made to the proxy 220 to retrieve the contents of a home web page 308 (such as an HTML page) from a web site 180, the HTML page and the contents of all references comprise a cache object group. Likewise if the request is directed to retrieving a file using, e.g., the file transfer protocol (FTP), the cache object is the file to be retrieved. Characterization of a cache object thus generally applies to the form of a cacheable data structure that is responsive to, and therefore defined by, a particular request.

Detailed Description Text (16):
A cache object manager process 350 controls the cache object store by implementing various aging and memory management algorithms, along with recording the number of data buffers that are allocable. For example, the cache object manager implements aging policies, such as a modified least recently used (LRU) algorithm described further herein. The cache object manager 350 also controls aging with respect to time of access and cost of re-acquisition of the data to thereby manage the full hierarchy (each level) of the cache store by, in part, determining when to remove select cache object groups. The cache object manager is preferably implemented as a state machine process that is executed, e.g., on the PMM platform; however, it will be apparent to those skilled in the art that the cache object manager may be implemented in hardware and may even reside on a different platform, e.g., in a shared memory model arrangement, as long as it has access to all of the cache memory.

Detailed Description Text (24):
The memory-level store is optimized by accessing the RAM devices through e.g., hardware index and displacement registers on a "natural" (4 K or 64 K) boundary, primarily because addressing is commonly mapped to such segmentation. The primary storage mechanism in RAM is a data buffer 312 preferably configured to accommodate a predetermined cache object size. The data buffers are used to "buffer" both data and metadata so that all disk operations occur through these buffers to reduce copies and transformations. Note that there are certain control structures that are not written to disk and thus do not use the data buffers.

First Hit Fwd Refs

☑ Generate Collection Print

L4: Entry 15 of 16 File: USPT Jun 4, 1996

DOCUMENT-IDENTIFIER: US 5524205 A
TITLE: Methods and apparatus for optimizing undo log usage

Abstract Text (1):
Each node in a data processing system contains at least one undo buffer and one least one redo buffer for insuring that any changes made to a section of a non-volatile storage medium, such as a disk, can be removed, if a transaction has not been committed, or can be recreated if the transaction has not been committed. The undo buffers each correspond to a different uncommitted transaction. The redo buffer contains the changes made to a copy of the section which is maintained in the memory.

Brief Summary Text (12):
Distributed data shipping systems, on the other hand, are decentralized so the same data can reside in the local memories of multiple nodes and be updated from these nodes. This results in multiple nodes logging actions for the same data.

Brief Summary Text (20):
The present invention avoids the problem of the prior art by ensuring that sufficient information from redo and undo buffers is maintained so that all changes of uncommitted transactions can be removed, the changes from the committed transactions can be recreated, and the storage of the undo buffers into undo logs can be minimized. Further efficiencies may be maintained by keeping a count of actions in a transaction as the actions are undone.

Brief Summary Text (21):
Specifically, in a data processing system including a plurality of nodes and a non-volatile storage medium divided into sections, the plurality of nodes making changes to the sections by way of transactions, each transaction comprising a series of changes made to at least one section by at least one node, and each transaction being committed if a record of the changes effected by that transaction as well as an indication of the completion of that transaction are reliably stored on the storage medium, and otherwise being uncommitted, a first one of the plurality of nodes comprises several elements. They include: a memory for holding a copy of at least one section; processing means, coupled to the memory, for making changes to the copy of the at least one section in the memory; at least one undo buffer containing a sequential list of the changes made by the processing means to the copy of the at least one section in the memory, each undo buffer corresponding to changes made by the first node to a different uncommitted transaction; a redo buffer containing a sequential list of changes made by the processing means in the corresponding node to the copy of the at least one section in memory; storing means, coupled to the memory, for storing the copy of the at least one section back into the storage medium; and log management means, coupled to the undo buffers and to the redo buffer, for selectively storing the portions of the undo buffers and redo buffer to the storage medium to ensure that the effects of all changes of uncommitted transactions can be removed and the effects of all changes of committed transactions can be recreated.

Detailed Description Text (7):
Another part of persistent storage is a backup tape system 150 which is referred to

as "archive storage." Archive storage is a term used generally to refer to the system storage used for information that permits reconstruction of the contents of persistent storage should the data in the persistent storage become unreadable. For example, should shared disk system 140 have a media failure, tape system 150 could be used to restore disk system 140. Archive storage frequently includes a magnetic tape system, but it could also include magnetic or optical disk systems as well.

Detailed Description Text (11):
As explained above, most data base systems use logs for recovery purposes. The logs are generally stored in persistent storage. When a node is updating persistent storage, the node stores the log records describing the updates in a buffer in the node's cache.

Detailed Description Text (12):
The preferred implementation of the present invention envisions three types of logs in persistent storage, but only two types of buffers in each node's cache. The logs are redo logs, or RLOGs, undo logs, or ULOGs, and archive logs, or ALOGs. The buffers are the redo buffers and the undo buffers.

Detailed Description Text (13):
An example of an RLOG is shown in FIG. 3, an example of a ULOG is shown in FIG. 4, and an example of an ALOG is shown in FIG. 5. The organization of a redo buffer is similar to the RLOG, and the organization of an undo buffer is similar to the ULOG.

Detailed Description Text (26):
Unlike RLOGs, each ULOG and undo buffer is associated with a different transaction. Thus ULOGs and their corresponding buffers disappear as transactions commit, and new ULOGs appear as new transactions begin. Other possibilities exist.

Detailed Description Text (30):
In FIG. 5, ALOG 500 is a preferred implementation of a sequential file used to store redo log records for sufficient duration to provide media recovery, such as when the shared disk system 140 in FIG. 1 fails. The RLOG buffers are the source of information from which ALOG 500 is generated, and thus ALOG 500 has the same attributes as RLOG 300.

Detailed Description Text (42):
When storing the updated block back to persistent storage, such as shared disk system 140, a Write-Ahead Log (WAL) protocol is used. The WAL protocol requires that the redo and undo buffers be written to the logs in shared disk system 140 before the blocks are written thereto. This ensures that the information necessary to repeat or undo the action is stably stored before changing the persistent copy of the data.

Detailed Description Text (45):
Thus, the WAL protocol is a necessary condition for an unbroken sequence of logged actions. It is also a sufficient condition with respect to block updates. When a block moves from one node's cache to another, the WAL protocol forces the RLOG records for all prior updates to the blocks to be changed by the committing transaction. "Forcing" means ensuring that the records in a nodes cache or buffer are stably stored in persistent storage.

Detailed Description Text (111):
ALOG truncation uses RLOG checkpoints. An RLOG checkpoint determines a safe point which permits the truncation of the RLOG as of the time of the checkpoint. This is because all versions of the data in persistent storage are more recent than this safe point, or else the point would not be safe.

Detailed Description Text (124):

One disadvantage of separate ULOGs and RLOGs on a redo log is that two logs must be forced when a block with uncommitted data is written to persistent storage in order to satisfy the WAL protocol. In general, however, writing blocks with uncommitted data to persistent storage should be sufficiently infrequent that the separation of logs provides a net gain, even in performance.

Detailed Description Text (133):
Actions end up on the ULOG for two reasons: either the WAL protocol forces a buffer record to the ULOG because the block was written to persistent storage, or the writing of the ULOG for WAL enforcement results in the writing of preceding ULOG records and, in some cases, following ULOG records that are in the undo buffer.

Detailed Description Text (137):
With the present invention, the use of the ULOG can be optimized by making sure that the contents of an undo log buffer are written to a ULOG only when necessary. In general, the undo buffer need only be stored to a ULOG when a block containing uncommitted data from a current transaction is written to persistent storage. If the transaction has been committed, there will be no need to undo the updates in the transaction, and thus the undo buffer can be discarded.

Detailed Description Text (139):
If the block to be written contains uncommitted data (step 1010), then the redo buffer needs to be written to the RLOG in the persistent storage, and any undo buffers are written to ULOGs in the persistent storage (step 1020).

Detailed Description Text (140):
After writing the redo buffers to the RLOG and the undo buffers to the ULOGs (step 1020), or if the blocks did not contain uncommitted data (step 1010), the block is written to the persistent storage (step 1030). This is in accordance with the WAL protocol.

Detailed Description Text (141):
Thus, the undo buffers are only written if there is uncommitted data to be stored. Each time a transaction commits, the corresponding undo log buffer can be discarded since it need not ever be written to the persistent storage. Furthermore, the ULOG itself for the transaction may be discarded as undo is now never required.

Detailed Description Text (152):
UNDO.sub.-- DATA attribute 1150 describes the nature of the action to be undone and provides enough information for the action to be undone after its associated original action has been incorporated into the block state. The value for the UNDO.sub.-- DATA attribute 1150 comes from the corresponding undo record stored either in a ULOG or in an undo buffer.

Detailed Description Text (178):
FIG. 14 shows a procedure 1400 for Block Update operation. First, the required concurrency control required is performed to lock the block for update (step 1410). The block is then accessed from persistent storage if it is not already in cache (step 1420). The indicated transaction is then performed upon the version of the block in cache (step 1430). Next, the block's DSI is updated with the ASI for the action (step 1440). Then, both RLOG and ULOG records are constructed for the update and are posted to their appropriate buffers (step 1450). The LastLSNs 950 (FIG. 9) are updated appropriately (step 1460). Then the NEXT 1250 value is set to the ULOG LSN of the undo record for this action (step 1470).

Detailed Description Text (180):
FIG. 15 contains a flow diagram 1500 for a Block Write operation if the block contains uncommitted data. First the WAL protocol is enforced (step 1510). Specifically, prior to writing the block to persistent storage, all undo buffers are written up to the corresponding LastULSN 958 (FIG. 9) for the block, and the

RLOG buffer is written up to the LastRLSN 955 (FIG. 9). For each transaction identified in the LastULSNs for the block, set HIGH for these transactions to the RLOG LSN values in the RLSN attributes of the undo records identified by the LastULSN attributes from the Dirty Blocks Table. Each LastULSN must identify both a transaction via a TID and a ULOG LSN. For these logs, there are times when no writing need be done because these records have already been written.

Detailed Description Text (207):
For each transaction, generated undo records are stored in the transaction's ULOG buffer. These undo records, plus those on its ULOG and its CLRs, ensure that an active transaction can be rolled back. Hence, at the end of the redo phase, all necessary undo log records will exist.

Detailed Description Text (210):
First all active transactions (but not prepared transactions) in Active Transactions table 1200 are rolled back. Undo processing proceeds exactly as in rolling back explicitly aborted transactions, with one exception. Some undo records might be present both in an undo buffer, where they were regenerated during redo, and in a ULOG in persistent storage. These duplicate undo records can be detected and ignored. This can be encapsulated in a routine to get the next undo record, so that the remainder of the code to undo transactions active at time of crash can be virtually identical to the code needed to undo a transaction when the system is operating normally. Redundant ULOG records among these sources can be eliminated because all undo records are identified by the LSN of the RLOG record to which they apply.

Other Reference Publication (5):
Crus, "Data Recovery in IBM Database 2," IBM Systems Journal, vol. 23, No. 2, 1984, pp. 178-188.

Other Reference Publication (8):
Lindsay, et al., "Notes on Distributed Databases," Research Report, (1979), pp. 1-57.

Other Reference Publication (15):
Kohler, W., "Overview of Synchronization and Recovery Problems in Distributed Databases", Fall COMPCON 80, Sep. 23-25, 1980, pp. 433-441.

CLAIMS:

1. A data processing recovery apparatus comprising:

a redo buffer containing a set of redo records, said redo buffer including information for committed and uncommitted transactions;

an undo buffer containing a set of undo records, said undo buffer including information only for an uncommitted transaction, said undo records being aggregated in said undo buffer separately from said redo records in said redo buffer; and

a log management routine for starting an uncommitted transaction, recording redo records corresponding to said uncommitted transaction in said redo buffer, recording undo records for said uncommitted transaction in said undo buffer, committing said transaction, storing said redo records corresponding to said committed transaction from said redo buffer to persistent storage, and for separately discarding said undo records corresponding to said committed transaction from said undo buffer while retaining said redo records in said redo buffer.

4. The data processing recovery apparatus as claimed in claim 2 further including a means for storing the contents of said undo buffer in said persistent storage prior to storing changes from corresponding uncommitted transactions.

5. A method for data processing recovery comprising the steps of:

providing a redo buffer containing a set of redo records, said redo buffer including information for committed and uncommitted transactions;

providing an undo buffer containing a set of undo records, said undo buffer including information only for an uncommitted transaction, said undo records being aggregated in said undo buffer separately from said redo records in said redo buffer;

starting an uncommitted transaction;

recording redo records corresponding to said uncommitted transaction in said redo buffer;

recording undo records for said uncommitted transaction in said undo buffer;

committing said transaction;

storing said redo records corresponding to said committed transaction from said redo buffer to persistent storage; and

separately discarding said undo records corresponding to said committed transaction from said undo buffer while retaining said redo records in said redo buffer.

Freeform Search

Database:
- US Pre-Grant Publication Full-Text Database
- [✓ US Patents Full-Text Database]
- US OCR Full-Text Database
- EPO Abstracts Database
- JPO Abstracts Database
- Derwent World Patents Index
- IBM Technical Disclosure Bulletins

Term: L2 and buffer$

Display: 50 **Documents in Display Format:** FRO **Starting with Number** 1

Generate: ○ Hit List [✓ Hit Count] ○ Side by Side ○ Image

Search Clear Interrupt

Search History

DATE: Saturday, January 24, 2004 Printable Copy Create Case

Set Name side by side	**Query**	**Hit Count**	**Set Name** result set
	DB=USPT; PLUR=YES; OP=OR		
L4	L2 and buffer$	16	L4
L3	L2 and (permanent near memory)	0	L3
L2	L1 and ((plurality or multiple) near memory)	28	L2
L1	database and (persistent near data)	607	L1

END OF SEARCH HISTORY